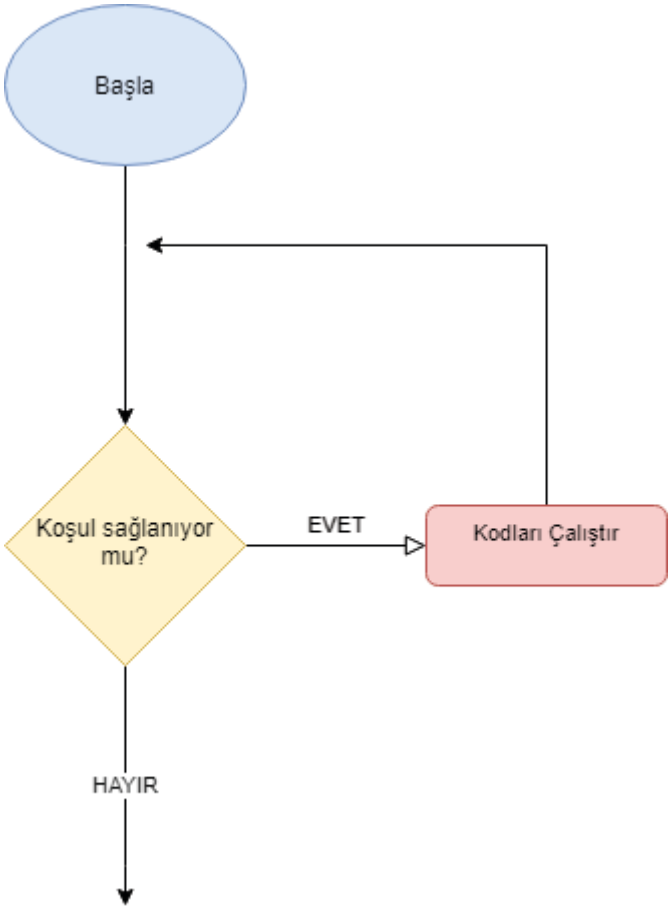
MODÜL 6

DÖNGÜ YAPILARI

Şekil 6.1: *Bölümle ilgili örnek uygulamalara karekoddan ulaşabilirsiniz.*

Bu bölüme kadar yapılan uygulamalar tek seferde çalışmaktadır. Örnek olarak 50 sayının ortalamasını almak istendiğinde, 50 sayıyı kullanıcıdan alarak her birini bir değişkene atamak, sonra da bu sayıların ortalamasının bulunması gerekiyordu. 50 sayı değil de bin adet sayının ortalamasını bulmak istersek bu iş daha da zorlaşacaktır.

Döngüler, istenen kodların belirli sayı veya koşullar sağlandığı sürece tekrar tekrar çalıştırılması temeline dayanır. Burada tekrarlama işlemi belirli sayıda olursa **for döngü yapısı**, belirli koşullara bağlı tekrar söz konusu ise **while döngü yapısı** tercih edilir. Örnek verilecek olursa her sabah güneş doğar ve her akşam güneş batar. Bu işlem süreklilik arz etmektedir.



Şekil 6.2: *Döngü Yapısı*

Yukarıdaki şekilde koşul sağlandığı sürece döngü devam edecektir. Ne zaman ki koşul şartı gerçekleşmezse o durumda döngüden çıkılacaktır.

Python'da **while** ve **for** döngüleri olmak üzere iki tür döngü bulunur.

6.1. While Döngüsü

While döngüsü, koşul gerçekleştiği sürece çalışan bir döngü çeşididir. Genellikle döngünün kaç defa çalışacağı belirli değilse while döngüsü tercih edilir. Ancak koşullar verilerek de while döngüsünün belirli sayıda çalışması sağlanabilir. Döngülerde koşullu ifadelerde olduğu gibi blok yapısı kullanılmaktadır. while ifadesinden sonra koşul durumu yazılır, ardından iki nokta işareti konularak alt satıra geçilir. Koşul durumu sağlandığı sürece çalışacak kodlar bir blok içeriden çalışır.

while (koşul durumu):

1. adım
2. adım
3. adım

.

.

Bu durum bir örnekle incelenmek istenirse,

Örnek 1

```
# şartın başlangıç değeri
sayac=1
#sayac 6 dan küçük olduğu sürece
while sayac<6:
    print("merhaba dünya")
    sayac=sayac+1
merhaba dünya
merhaba dünya
merhaba dünya
merhaba dünya
merhaba dünya
```

Örnek 1'de sayaç isimli değişkenin değeri 1'den başlamış, yine değişken değeri 6'dan küçük olduğu sürece konsol üzerinde "merhaba dünya" yazılır. Sayacın değeri 1 arttırılarak döngünün başına döner ve sayaç değeri 6'ya eşit olana kadar bu durum devam eder. Aynı işlemi ekrana sayaç isimli değişkenin değeri yazılarak yapmak istenirse,

Örnek 2

```
sayac=1
while sayac<6:
    print(sayac)
    sayac=sayac+1
1
2
3
4
5
```

6.1.1. Döngünün Kapsamı

Hatırlanacağı üzere döngü koşulunun sağlandığı sürece daha içteki bloklarda bulunan kodların çalışacağı belirtilmişti. Döngü bittiği zaman Python bir üstteki bloğa dönerek çalışmasına devam eder.

Örnek 3

```
# şartın başlangıç değeri
sayac=1
#sayac 6 dan küçük olduğu sürece
while sayac<6:
    print(sayac)
    sayac=sayac+1
#döngü bittiği zaman
print("döngü sonlandı")
1
2
3
4
5
döngü sonlandı
```

while döngüsü ile çalışırken sık yapılan hatalardan birisi döngü içerisinde koşulu sağlayan değişkenin değerini arttırma işleminin unutulmasıdır. Bu durumda koşul sürekli sağlanacağı için döngü sürekli çalışır ve dışarıdan bir müdahale ile sonlandırılması gerekir.

Örnek 4

```
sayac=1
while sayac<6:
    print(sayac)
```

Yukarıdaki örnekte bilinçli olarak sayaç isimli değişkenin değeri arttırılmamıştır. Uygulamayı çalıştırıldığında görüleceği üzere program hiç durmadan çalışacak ve ekrana sürekli 1 değeri yazacaktır. Bu işlem sırasında klavyeden "ctrl + c" tuşuna basarak Python'a kesme gönderebilir ve uygulama sonlandırılabilir.

Örnek 5

while döngüsü kullanarak 1-100 arasındaki (100 dâhil) çift sayıları bularak ekrana yan yana yazan programı yazınız.

```
a=1
while a<=100:
    if a%2==0:
        print(a,end=",")
    a=a+1
2,4,6,8…..,100
```

Örnek 6

while döngüsü kullanarak 1 – 100 arasındaki sayıların toplamını bulan programı yazınız.

```
toplam=0
i=1
while i<=100:
    toplam=toplam+i
    i=i+1
print("sayıların toplamı",toplam)
sayıların toplamı 5050
```

Örnek 6'da yapılan işlemde toplam isimli bir değişken oluşturularak başlangıç değeri 0 olarak belirlenmiştir. Çünkü toplam değerini hesaplarken toplam=toplam+i şeklinde bir işlem yapılmaktadır. Eğer toplam tanımlı olmasaydı, tanımlanmamış_değer=tanımlanmamış_değer+sayı şeklinde bir işlem yapılmaya çalışacak ve hata verecekti. Yani bir değişken ile işlem yapılmadan önce tanımlanması gerekmektedir.

6.1.2. While True – Break İfadeleri ve Sonsuz Döngüler

Program yazarken bazen döngünün ne zaman sonlanacağı bilinmeyebilir. Örnek olarak bir markette müşterilerin alışveriş yaparak sepetlerini doldurdukları ve sepette kaç adet ürün olduğu bilinmeyebilir. O müşteriye ait tüm ürünler barkod okuyucu ile okutulmalı ve toplam tutar hesaplanmalıdır. İşte bu gibi belirsiz durumlar için while döngüsü ile beraber **True** ifadesi ya da benzer yapılar kullanılabilir. **Break** ifadesi ise döngü sürekli çalışırken istenilen bir anda döngüden çıkmak için kullanılır.

Örnek 7

```
i=1
while True:
    print(i)
    i+=1
    if i==6:
        break
print("döngü sonlandı")
1
2
3
4
5
döngü sonlandı
```

while döngüsü ile kullanılan True ifadesinin yerine farklı kullanımlarla da karşılaşılabilir. Örneğin, bir alışveriş yapıldığını ve sürekli ürün girişi yapıldığını düşünülürse "q" harfi girilene kadar yapılan alışverişler listeye eklensin, eğer "q" harfi girilirse döngüyü sonlandırılsın.

Örnek 8

```
liste=[]
while 1:
    ürün=input("ürün adı giriniz:")
    if ürün=="q":
        break
    liste.append(ürün)
print("girdiğiniz meyveler:",liste)
ürün adı giriniz:elma
ürün adı giriniz:armut
ürün adı giriniz:peynir
ürün adı giriniz:su
ürün adı giriniz:q
girdiğiniz meyveler: ['elma', 'armut', 'peynir', 'su']
```

Örnek 9

while döngüsü kullanarak sayı tahmin oyunu yapınız. Kullanıcıdan 1-100 arası bir sayı istenmektedir. Girilen sayı, tahmin edilen sayıdan büyükse daha küçük bir sayı girmesi, büyükse daha küçük bir sayı girmesi istensin. Kullanıcı sayıyı bulana kadar bu işlem tekrar etsin. Ayrıca bir sayaç eklenerek kaç defa da tahmin ettiğini bulunuz.

```
Sayi=45
sayaç=0
print("1-100 arası bir sayı tuttum tahmin et")
while 1==1:
    sayaç+=1
    cevap=int(input("1-100 arası bir sayı girin: "))
    if cevap>sayi:
        print("daha küçük bir sayı girmelisin")
    elif cevap<sayi:
        print("daha büyük bir sayı girmelisin")
```

```
    else:
        print("tebrikler tuttuğum sayıyı bildin")
        break
print("tebrikler {} seferde sayıyı bulabildin".format(sayaç))
1-100 arası bir sayı tuttum tahmin et
1-100 arası bir sayı girin: 50
daha küçük bir sayı girmelisin
1-100 arası bir sayı girin: 40
daha büyük bir sayı girmelisin
1-100 arası bir sayı girin: 45
tebrikler tuttuğum sayıyı bildin
tebrikler 3 seferde sayıyı bulabildin
```

Yine burada **while True** yapısına benzer bir yapı kullanarak while 1==1: şeklinde bir yapı ile sonsuz döngü oluşturulmuştur.

Örnek 10

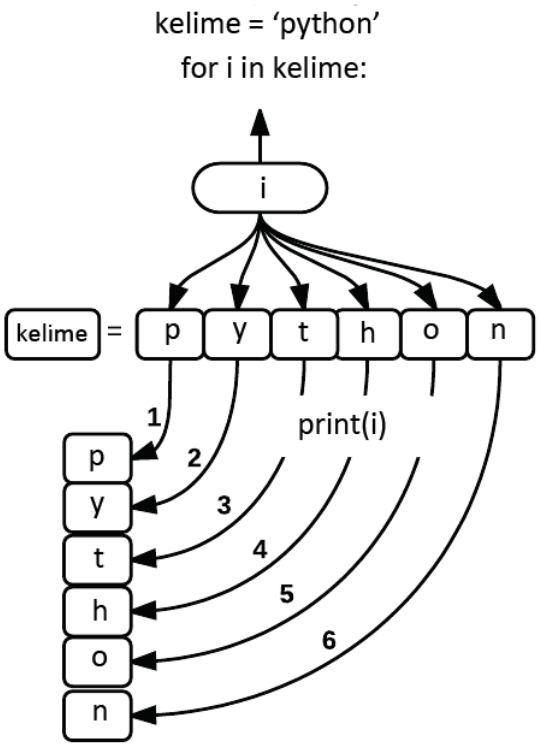
Girilen sayının faktöriyelini hesaplayan programı yazınız.

```
i=1
f=int(input("faktöriyeli alınacak sayıyı giriniz: "))
sonuc=1
while i<=f:
    sonuc=sonuc*i
    i+=1
print(sonuc)
faktöriyeli alınacak sayıyı giriniz: 5
120
```

Örnek 10'da i isimli değişkeni 1 değerinden başlayarak while döngüsünü i değeri girilen sayıdan küçük olduğu sürece çalıştırılmış, daha sonra i değeri arttırarak sonuç isimli değişkenle çarpma işlemi yapılmış, i değeri f değerine eşit olduğunda döngüden çıkılarak, sonuç isimli değişkenin değeri gösterilmiştir.

6.2. For Döngüsü

For döngüsü, Python'da genellikle döngünün **tekrar sayısı programcı tarafından belirlenmiş veya öngörülmüş ve** belli ise kullanılır. Hatırlanacağı üzere **while** döngüsü ile sonsuz döngüler yapılabiliyor ve istenilen bir anda döngüden çıkılabiliyordu. **For** döngüsü daha çok belirli sayıdaki işlemi gerçekleştirmek için kullanılır. Bunun yanında Python'da for döngüsünün **iterasyon** denilen önemli bir özelliği bulunmaktadır. **İterasyon** işlemi sayesinde karakter dizileri ve listeler üzerinde gezinme işlemi, yani ilk elemandan son elemana kadar işlem yapabilmektedir. **For** döngüsü kullanmak için **"in"** işlecinden faydalanmak gerekmektedir.



Şekil 6.3: İterasyon işlemi

6.2.1. in İşleci

in işleci bir değerin, bir liste ya da karakter dizisi içerisinde olup olmadığını kontrol eder. Önce karakter dizisi içinde bir karakter olup olmadığına bakar. Eğer değer karakter dizisi içerisinde varsa **True**, yoksa **False** değeri döndürecektir. Aşağıdaki kodları etkileşimli kabuk üzerinde çalıştırabilirsiniz.

Örneğin:

"p" in "python"

```
>>> True
```

"a" in "python"

```
>>> False
```

aynı işlemi listeler üzerinde de yapılabilir.

```
"a" in "python"
>>> False
liste=[1,2,3,4,5,6]
3 in liste
>>> True
10 in liste
>>> False
```

6.2.2. Karakter Dizileri Üzerinde İterasyon İşlemi

Python'da **for** döngüsü ile karakter dizileri üzerinde kolaylıkla iterasyon işlemi yapılabilir.

Örnek 11

```
isim="Mustafa"
for i in isim:
    print(i,end=",")
M,u,s,t,a,f,a,
```

Örnek 11'de yapılan işlemle isim adlı değişken üzerinde ilk karakterden son karaktere kadar tüm değerleri hiçbir harf kalmayana kadar sırayla i değişkenine atayarak ekrana yazdırılmıştır.

Aynı işlemi while döngüsü ile yapılmak istenirse,

Örnek 12

```
isim="Mustafa"
i=0
while i<len(isim):
    print(isim[i],end=",")
    i=i+1
M,u,s,t,a,f,a,
```

Örnek 12'de **while** döngüsü ile karakter dizileri üzerinde işlem yapılmak istendiğinde hem daha fazla kod yazılması hem de daha karışık bir yapı kullanılması gerekmektedir. Python'da genellikle listeler veya karakter dizileri üzerinde işlem yapılmak istenildiği zaman yani iterasyon yapılacağı zaman **for** döngüsü kullanılmaktadır.

Örnek 13

Bir cümle içerisinde geçen bir harfin kaç defa geçtiğini bulunuz.

```
yazi="Python üst düzey basit sözdizimine sahip, öğrenmesi oldukça kolay,
modülerliği, okunabilirliği desktekeyen, platform bağımsız nesne yönelimli
yorumlanabilir bir script dilidir."
harf="a"
sayac=0
for i in yazi:
    if i=="a":
        sayac=sayac+1
print("cümle içerisinde geçen a harfi sayısı: ",sayac)
cümle içerisinde geçen a harfi sayısı:  9
```

Örnek 14

Bir cümle içerisinde geçen sesli harfleri bulan programı yazınız.

```
yazi="Python üst düzey basit söz dizimine sahip, öğrenmesi oldukça kolay,
modülerliği, okunabilirliği destekleyen, platform bağımsız nesne yönelimli
yorumlanabilir bir script dilidir."
sesli="aeıioöuü"
for i in yazi:
  if i in sesli:
    print(i,end=",")
```

```
o,ü,ü,e,a,i,ö,i,i,i,e,a,i,ö,e,e,i,o,u,a,o,a,o,ü,e,i,i,o,u,a,i,i,i,i,e,e,e,e,a,o,a,ı,
ı,e,e,ö,e,i,i,o,u,a,a,i,i,i,i,i,i,i,
```

Örnek 15

İki farklı karakter dizisi belirleyerek, birinci de olup, diğerinde olmayan karakterleri bulunuz.

```
cumle1="Python üst düzey basit sözdizimine sahip, öğrenmesi oldukça kolay,
modülerliği, okunabilirliği destekleyen, platform bağımsız nesne yönelimli
yorumlanabilir bir script dilidir."
cumle2=" Python interaktif yani etkileşimli bir programlama dilidir."
for i in cumle2:
    if not i in cumle1:
        print(i,end=",")
```

```
ş,ğ,
```

Örnek 16

Kullanıcı tarafından girilen bir karakter dizisi içerisinde geçen sesli ve sessiz harfleri ayrı ayrı listelere atayan programı yazınız.

```
sesli_harfler = "aeıioöuü"
sessiz_harfler = "bcçdfgğhjklmnprsştvyz"
sesliler=""
sessizler=""
a=input("bir metin giriniz")
for i in a:
    if i in sesli_harfler:
        sesliler=sesliler+i
    if i in sessiz_harfler:
        sessizler=sessizler+i
print("sesli harfler",sesliler)
print("sessiz harfler",sessizler)
bir metin giriniz Python üst düzey basit söz dizimine sahip, öğrenmesi oldukça
kolay, modülerliği, okunabilirliği destekleyen, platform bağımsız nesne yönelimli
yorumlanabilir bir script dilidir.
sesli harfler oüüeaiöiiieaiöeeiouaoaoüeiiouaiiiieeeeaoaııeeöeiiouaaiiiiiii
sessiz harfler ythnstdzybstszdzmnshpğrnmsldkçklymdlrlğknblrlğdsktkynpltfrmbğmsznsnynl
mlyrmlnblrbrscrptdldr
```

6.2.3. Listeler Üzerinde İterasyon İşlemi

Python'da karakter dizilerinde yapılan işlem gibi listeler üzerinde de iterasyon işlemi yapılabilir. Örnek olarak elimizde bir liste olduğunu ve içerisinde sayısal ifadeler (integer tanımlanmış) olduğunu düşünelim. Bu değeri toplayabilir, ortalamasını bulabilir ya da farklı bir listeye atayabilirsiniz. Kullanım şekli itibariyle karakter dizilerinde yapılan işlemlerin aynısı yapılabilir.

Örnek 17

İçerisinde sayısal değerler olan bir listedeki değerlerin karesini alarak başka bir listeye atayınız.

```
sayılar = [1,2,3,4,5]
kareler = []
for i in sayılar:
    kareler.append(i*i)
print(kareler)
[1, 4, 9, 16, 25]
```

Örnek 18

Bir liste içerisinde bulunan değerlerin ortalamasını bulun.

```
sayılar = [1,2,3,4,5,6,7,8,9]
toplam=0
for i in sayılar:
    toplam=toplam+i
print("sayıların ortalaması: ",toplam/len(sayılar))
sayıların ortalaması:  5.0
```

Örnek 19

Listedeki değerlerden 3'ün katları olan sayıları ekrana yazdırın.

```
sayılar = [5,8,12,17,25,36,41,49,60,72]
for i in sayılar:
    if i%3==0:
        print(i,end=",")
12,36,60,72,
```

Örnek 20

İç içe listeler üzerinde gezinme işlemi yapılabilir. [[3,4],[7,8],[10,11],[14,15]] şeklinde bir liste olduğunu varsayalım. Liste elemanları üzerinde klasik bir şekilde gezinme işlemi yapılmak istenirse:

```
liste = [[3,4],[7,8],[10,11],[14,15]]
for i in liste:
    print(i,end=",")
 [3, 4],[7, 8],[10, 11],[14, 15],
```

Listeye erişildi ancak alt listelere erişilebilmesi için aşağıdaki gibi bir yapının kullanılması gerekmektedir.

Örnek 21

```
liste = [[3,4],[7,8],[10,11],[14,15]]
for i,j in liste:
    print(i,end=",")
    print(j,end=",")
3,4,7,8,10,11,14,15,
```

Örnek 22

Ya da iç içe for döngüsü kullanarak aynı işlem gerçekleştirilebilir.

```
liste = [[3,4],[7,8],[10,11],[14,15]]
for i in liste:
    for j in i:
        print(j,end=",")
3,4,7,8,10,11,14,15,
```

6.2.4. range Fonksiyonu ile For Döngüsü Kullanımı

Python'da **for** döngüsüyle belirli değerler arasında döngü kurmak istenirse **range** fonksiyonu kullanılmalıdır. **range** fonksiyonu bir sayı dizisi oluşturur ve bu sayede oluşturulan sayı dizisi üzerinde **for** döngüsünün iterasyon yapması sağlanır.

Örnek 23

```
print(*range(10))
0 1 2 3 4 5 6 7 8 9
```

range fonksiyonu, girilen aralık arasında integer değerler oluşturur. Örnek 23'te aralık belirtilmediği için başlangıç değeri 0 alınmıştır. Başlangıç değeri verilerek girilen değerler arasında sayı dizisi oluşturulması sağlanabilir.

Örnek 24

```
print(*range(5,20))
5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
```

Ayrıca range fonksiyonuna üçüncü bir parametre verilerek atlama değeri de verilebilir.

Örnek 25

```
print(*range(1,20,3))
1 4 7 10 13 16 19
```

Örnek 26

for döngüsü ile girilen sayıya kadar olan sayıların toplamını bulunuz.

```
toplam=0
for i in range(20):
    toplam=toplam+i
print("girdiğiniz sayıların toplamı:",toplam)
girdiğiniz sayıların toplamı: 190
```

Örnek 27

20'den geriye doğru 0'a kadar olan sayıları ekrana yazdırınız.

```
for i in range(20,0,-1):
    print(i,end=",")
20,19,18,17,16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,
```

Örnek 28

100'e kadar 5'in katları olan sayıyı bulunuz.

```
for i in range(0,100,5):
    print(i,end=",")
0,5,10,15,20,25,30,35,40,45,50,55,60,65,70,75,80,85,90,95,
```

Örnek 29

Kullanıcıdan satır ve sütun sayısı değerlerini alarak sayıları tablo şeklinde yazan programı yazınız.

```
a=int(input("tablonun satır uzunluğunu giriniz"))
b=int(input("tablonun sütun uzunluğu giriniz"))
for i in range(1,a+1):
     for j in range(1,b+1):
         print(j,end=" ")
     print()
tablonun satır uzunluğunu giriniz5
tablonun sütun uzunluğu giriniz4
1 2 3 4
1 2 3 4
1 2 3 4
1 2 3 4
1 2 3 4
```

Örnek 29

Girilen bir metindeki sesli harf, sayı ve özel karakterlerin sayısını bulan programı yazınız.

```
sesli_harfler=["a","e","ı","i","o","ö","u","ü"]
rakamlar="1234567890"
ozel_harf=["@","!","&","?"]
ozel_harf_sayısı,rakam_sayısı,sesli_sayısı=0,0,0
kelime=input("lütfen incelemek için bir metin giriniz: ")
for harf in kelime:
    if harf in sesli_harfler:
        sesli_sayısı+=1
    if harf in rakamlar:
        rakam_sayısı+=1
    if harf in ozel_harf:
        ozel_harf_sayısı+=1
```

```
print("girdiğiniz metinde {} adet sesli harf {} adet rakam ve {} adet özel karakter bulunmaktadır. ".format(sesli_sayısı,rakam_sayısı,ozel_harf_sayısı) )
lütfen incelemek için bir metin giriniz: girilen metindeki sesli harf sayı ve özel karakterli bulan program123456!@&
girdiğiniz metinde 23 adet sesli harf 6 adet rakam ve 3 adet özel karakter bulunmaktadır.
```

6.3. Continue İfadesi

Hatırlanacağı üzere **break** ifadesi döngünün dışına çıkılmasını sağlamaktadır. Döngülerde kullanılan **continue** ifadesi, döngünün baştan sona kadar çalışmasını engellemeyen ancak belirli durumlar sağlandığında o adımı atlamamızı sağlayan yapılardır. Döngü sona ermez ancak verilen koşulun sağlanması durumunda döngüyü direk başa alır.

Örnek 30

```
for i in range(1,6):
    if i ==2 or i==4:
        continue
    print(i)
1
3
5
```

Görüldüğü üzere Python, 2 veya 4 değerlerini görünce döngünün başına gitmiş ve alt satırdaki ifadeler çalıştırılmamıştır.

Örnek 31

Aynı işlem **while** döngüsü ile yapılmak istenirse,

```
i=0
while i < 5:
    i=i+1
    if i == 2 or i==4:
        continue
    print(i)
1
3
5
```

while döngüsü ile **continue** deyimi kullanırken döngü, sonsuz döngüye girebilir. Eğer i=i+1 ifadesi **continue** deyiminin altında kullanılırsa i değeri sürekli 2 olarak kalır. Değişkenin değeri artmaz ve uygulama sonsuz döngüye girer. Örnek 32'de hatalı kullanım şekli gösterilmiştir.

Örnek 32

```
i=0
while i < 5:
    if i == 2 or i==4:
        continue
     i=i+1
    print(i)
```

6.4. İç İçe Döngüler

Döngü bloklarının içerisine kodlar eklenebileceği gibi, koşul durumları hatta başka bir döngü koymak bile mümkündür. Bir döngü yapısının içine başka bir döngü yapısının yerleştirilmesi ile elde edilen yapıya **iç içe döngü (nested loop)** adı verilir.

Örnek 33

```
dersler = ["Ders 1", "Ders 2", "Ders 3"]
konular = ["Konu 1", "Konu 2", "Konu 3"]
for x in dersler:
    for y in konular:
        print(x, y)
Ders 1 Konu 1
Ders 1 Konu 2
Ders 1 Konu 3
Ders 2 Konu 1
Ders 2 Konu 2
Ders 2 Konu 3
Ders 3 Konu 1
Ders 3 Konu 2
Ders 3 Konu 3
```

Örnek 34

İç içe **while** döngüsü kullanarak, i ve j olarak tanımlanan iki değişkenin değerlerini ekrana yazdırınız.

```
i=1
while i<4:
    j=1
    while j<4:
        print("i nin değeri: {} j nin değeri: {} ".format(i,j))
        j=j+1
    i=i+1
i nin değeri: 1 j nin değeri: 1
i nin değeri: 1 j nin değeri: 2
i nin değeri: 1 j nin değeri: 3
i nin değeri: 2 j nin değeri: 1
i nin değeri: 2 j nin değeri: 2
i nin değeri: 2 j nin değeri: 3
i nin değeri: 3 j nin değeri: 1
i nin değeri: 3 j nin değeri: 2
i nin değeri: 3 j nin değeri: 3
```

6.5. Bölüm Sonu Örnekleri

1. Girilen sayının asal olup olmadığını bulan programı yazınız.

2. Kullanıcıdan değer alarak ağaç şekli çizen programı yazınız.

3. Bir listede bulunan sayıların en büyüğünü ve en küçüğünü bulan programı yazınız.

4. Elinizde bulunan iki adet listeyi birleştirerek 3. oluşturan programı yazınız.

5. Bir cümledeki boşlukları kaldıran programı yazınız.

Cevaplar

1.
```
a=int(input("bir sayı giriniz"))
asal=0
for i in range (2,a):
    if a%i==0:
        asal+=1
if asal==0:
    print("girdiğiniz sayı asal")
else:
    print("girdiğiniz sayı asal değil")
```

2.
```
a=int(input("ağacın yükseliği"))
b=a
for i in range(1,a+1):
    print(b*" ",(2*i-1)*"*")
    b-=1
```

3.
```
liste=[5,9,7,1,2,3,6,4]
buyuk=0
kucuk=999
for i in range(len(liste)):
    if liste[i]>buyuk:
        buyuk=liste[i]
    if liste[i]<kucuk:
        kucuk=liste[i]
print("""girilen sayıların en büyüğü= {}
girilen sayıların en küçüğü= {}""".format(buyuk,kucuk))
```

4.
```
liste1=[1,2,3,4,5,6,7]
liste2=["python","java","c","c++","c#","pascal","cobol"]
liste3=[]
for i in range(len(liste1)):
    liste3.append((liste1[i],liste2[i]))
print(liste3)
```

5.
```
a="m e r h a b a"
for i in a:
    if i!=" ":
      print(i,end="")
```